FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Dynamic Cruise Control

**Rui Pedro Barbosa Leão da Cunha**


Master's in Electrical Engineering

Supervisor: Sérgio Reis Cunha (PhD)

June 29, 2015

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

**MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES** | **2014/2015**

A Dissertação intitulada

"Dynamic Cruise Control"

foi aprovada em provas realizadas em 24-07-2015

**o júri**

Joaquim José de Amaral Vieira e Costa

Presidente Professor Doutor Joaquim José de Amaral Vieira e Costa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

Armando Carlos D. da Rocha

Professor Doutor Armando Carlos Domingues da Rocha
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática
da Universidade de Aveiro

Sérgio Reis Cunha

Professor Doutor Sérgio Reis Cunha
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Rui Pedro Barbosa Leão da Cunha

Autor - Rui Pedro Barbosa Leão da Cunha

Faculdade de Engenharia da Universidade do Porto

© Rui Cunha, 2015

# Abstract

A prototype of a dynamic cruise control system is developed and tested on a car in a highway environment.

By allowing the cruise control to dynamically change the speed, it is possible to reduce fuel consumption as well as breaks usage, making driving more efficient, clean and comfortable.

Mapping techniques, based on terrain data collection, are developed to build topographic maps of roads , including speed limit zones. The resultant maps are then used to optimize speeds in order to reduce fuel consumption, respecting the average speed set by the driver. The dynamic speeds are then a result of an energy cost function optimization.

Measuring the fuel spent for the different speed control methods, we conclude that using the dynamic cruise control it's possible to achieve fuel savings up to 23.25% against the traditional cruise control.

# Resumo

É desenvolvido e testado num carro, em autoestrada, um protótipo de um cruise control dinâmico.

Permitindo ao cruise control modificar dinamicamente a velocidade, é possível reduzir o consumo de combustível assim como diminuir o uso dos travões, contribuindo para uma condução mais ecológica, mais confortável e menos dispendiosa.

São desenvolvidas técnicas de mapeamento topográfico das estradas e estabelecidas zonas de velocidade limitada, com base em recolha de dados do terreno. Os mapas resultantes são utilizados para otimizar velocidades com o objetivo de reduzir o consumo de combustível, respeitando uma velocidade média, selecionada pelo condutor. As velocidades dinâmicas são resultado de um processo de otimização de uma função custo de energia.

Medindo o consumo de combustível para os diferentes métodos de controlo, concluímos que, usando o cruise control de velocidade dinâmica, é possível economizar pelo menos 23.25% de combustível, comparativamente ao cruise control tradicional.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor Sérgio Reis Cunha for promptly accept my request to guide me in this project, and for his vast knowledge and experience that did constantly helped me to find shortcuts among difficult decisions and problems. To my friend Emanuel Fonseca for motivating me and helping with equations and useful advices. To my friend Vasco Sotomaior for helping me solving LaTeX problems and lending me sound and video equipment to make a presentation video. To my friend André Cardoso for his motivation, constant concern and for taking care of our farming project while I was focused on this thesis. To my family, especially my parents that always believed in me and supported my decisions and to my grandma who made a great effort, trying not to interrupt me as much, asking for help on taking care of the animals and in agriculture. And most of all, I want to thank my girlfriend Helena Guimarães for her patience and encouragement, for all the hours listening to me talking about things that she doesn't fully understand and for her tedious task of reading my paper and help with corrections.

Rui Cunha

*"Para baixo todos os santos ajudam,*
*para cima é que as coisas mudam."*

Portuguese popular saying

# Contents

# List of Figures

# List of Tables

# Abbreviations and Symbols

| | |
|---|---|
| ACC | Adaptive Cruise Control |
| AICC | Autonomous Intelligent Cruise Control |
| API | Application Programming Interface |
| CACC | Cooperative Adaptive Cruise Control |
| CC | Cruise Control |
| CFCW | Cooperative Forward Collision Warning |
| DAC | Digital to Analog Converter |
| DCC | Dynamic Cruise Control |
| ECC | Eco-Cruise Control |
| ECU | Engine Control Unit |
| I/O | Input/Output |
| MCU | Microcontroller Unit |
| PCB | Printed Circuit Board |
| PI | Proportional Integral controller |
| PID | Proportional Integral Derivative controller |
| RPC | Remote Procedure Call |
| V2V | Vehicle to Vehicle, vehicles use wireless communications to transmit public and private data |

# Chapter 1

# Introduction

This chapter contextualizes the subject of this dissertation with the present society, identifying the main goals to achieve. It ends with an overview of the document structure.

## 1.1 Context

Car companies are focused on optimizing fuel consumption by producing more efficient engines and using lighter building materials. Although mechanical improvements are of extreme importance, all its potential can be wasted when the final consumer is not able to make an optimal control of the accelerator pedal.

While our society is still dependent on limited fossil fuels, it's crucial to use them as less as possible.

## 1.2 Goals

Roads are uneven, and for the same speed, cars have different fuel consumption, whether on uphill or downhill. The main goal of this thesis is to build a cruise control (CC) system, for a car, that relies on topographic data to automatically adjust the speed in order to optimize fuel consumption. With this system the driver doesn't select a fixed speed, but an average speed. The optimization process should then respect the drivers selected average speed, making speed variations within a range around that selection.

Has the reference speed set by the driver is the average speed and the real speed will oscilate arround it, we call to this system dynamic cruise control (DCC).

## 1.3 Objectives and Functionalities

As stated in 1.2, a DCC system has to be developed. This system has some mandatory functionalities as follows:

- It has to be able to collect slope information from roads and store it on a database in order to make topographic maps, has they are a fundamental part to the fuel optimization.
- Has not all roads are known, the DCC has to operate in two modes. One where it acts has a traditional CC and other where it will run the algorithm to choose the best speed within a range, centered on the driver's selected speed.
- To ensure safety, the connector from the DCC device that will override the signal from the physical accelerator pedal, has to be able to switch to manual control when a pedal is pressed or if the control module is turned off by a pressure button.
- A user interface is needed to allow real-time interaction with the device.
- Has the car's velocity will not be constant, a LCD may be used to display information like the actual selected speed.
- The DCC mode has to ensure that the average speed of a trip is close to the user-selected reference speed and it should set minimum and maximum speed based on the reference speed and ensure that the speed limits will be respected.

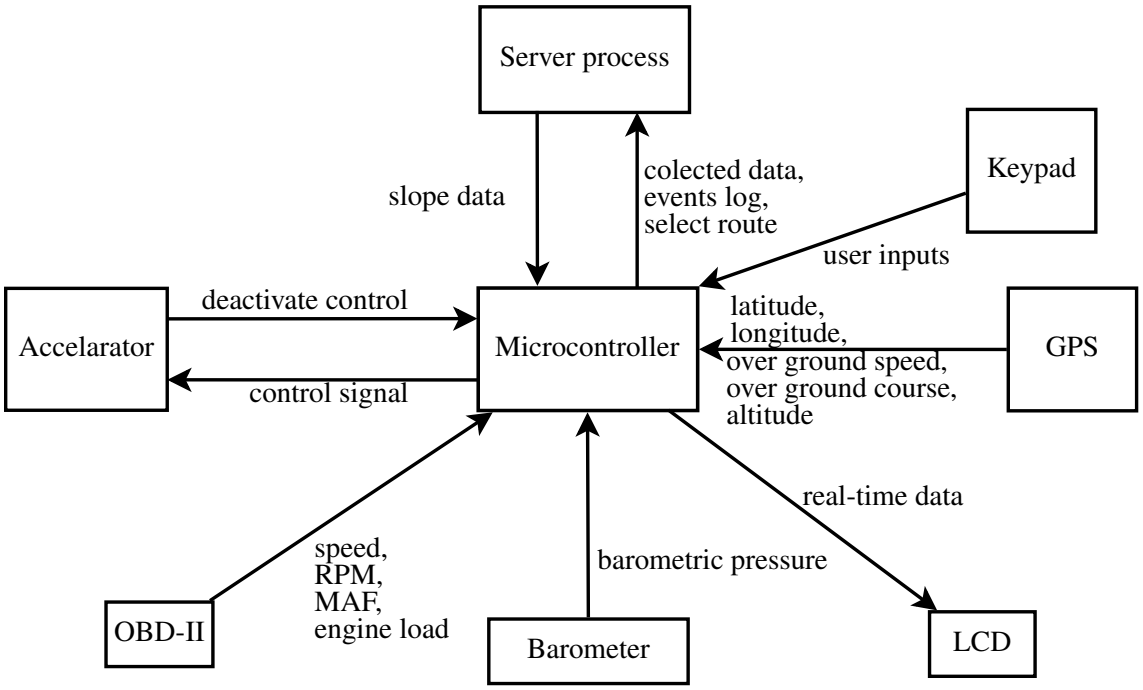



Figure 1.1: System overview.

## 1.4 Document Structure

Besides introduction, this document has four more chapters.

Chapter 2, describes the state of the art and recent cruise control developments and products are presented.

Chapter 3, addresses in detail the problems to be solved, the hardware configuration and data acquisition and processing methodologies.

Chapter 4, presents the data processing and analysis results.

Finally, chapter 5 is a results discussion of the developed project and an insight of improvements that could be made on a future work.

# Chapter 2

# State of the Art

This chapter briefly reviews the existing products and recent developments of CC technology.

## 2.1 Current Cruise Control Solutions

The traditional CC was first designed to keep a vehicle at constant speed. Nowadays it is able to change the reference speed over different transit scenarios.

### 2.1.1 Traditional Cruise Control

The traditional CC is a device that keeps a constant speed, selected by the driver, taking control over the throttle valve. If the breaking pedal is pressed, it deactivates the CC, so that the driver has full control over the vehicle.

This system is very simple, because it consists only on a proportional-integral-derivative controller (PID) or a proportional-integral controller (PI).

### 2.1.2 Adaptive Cruise Control

More recent cars are equipped with the Adaptive Cruise Control (ACC).

The big difference between the ACC and the conventional CC is that the system has an integrated radar able to detect and calculate the distance to a following vehicle. When this happens the previous selected cruise speed may be changed in order to keep a security distance to the following car and then if the road is clear, the velocity goes back to the user's selected [1].

This is the only device on the market that is able to change the selected cruise velocity.

### 2.1.3 BMW Dynamic Cruise Control

At the BMW official web site, they refer to their cruise control system as Dynamic Cruise Control. This system has nothing to do with this project. In fact the techniques used are in a way the opposite of this thesis, because they result in more fuel and breaks spending. They call it Dynamic Cruise Control, but it is more static than dynamic, because its main goal is to accurately keep the

reference speed and increase safety by introducing two new functions to the normal CC. The first is its ability to control the breaking system, so that the car can follow the reference speed even when on a downhill, where reducing the throttle to the minimum isn't enough and the speed would be higher than the reference. The second functionality is the *Curve Speed Limiter*, that detects road's curves and if needed, reduces the speed to offer a safer driving experience.

## 2.2 Recent Developments

Different CC systems have been constantly developed to actuate in autonomous cars and cars equipped with vehicle to vehicle communication (V2V).

### 2.2.1 Autonomous Intelligent Cruise Control

Autonomous Intelligent Cruise Control (AICC) is a great evolution on the CC technology.

The AICC is not only able to control the throttle, but it can also actuate on breaks and gearbox. [3, 4] This makes it possible to develop autonomous vehicles like the Google's Self-Driving Car and others from several car companies.

### 2.2.2 Cooperative Adaptive Cruise Control

Cooperative Adaptive Cruise Control (CACC) is a system designed to operate under V2V environment. The main goal is to avoid accidents, but there are other advantages being exploited, like optimize flows in heavy traffic or to reduce fuel consumption.

With the V2V technology, cars are able to communicate transit information to each other. So if some unexpected event happens, like an accident or a sudden traffic slow down, cars driving on that direction will be advised and the velocity can be changed by the CACC in order to avoid an accident.

Besides the security enhancement, there are some studies with traffic simulations where defined platoons of cars and algorithms are being tested to optimize the traffic flows. The goal here is to keep the traffic flow smooth which has been proven more secure and efficient. [5, 6]

Other studies focus on fuel efficiency. For this optimization, the system is aware of the traffic, so the velocity of each car is controlled not to save fuel on that individual car, but on a platoon of cars. [7, 8]

This developments are promising, but they are looking to a distant future. The V2V technology is still in a stage of development and all this techniques are only efficient on environments where all the cars on the road have a CACC installed, although there are also studies with groups of mixed cars with and without CACC.

### 2.2.3 Eco-Cruise Control

At the time the idea of this thesis came up, no pappers were available on the subject. However, during this project development, new articles were published entitled of Eco-Cruise Control (ECC), where the main idea is similar to the one on this thesis.

The most recent document is a patent by Ford Global Technologies, published on March of 2015 [9]. Ford's patent mention an Eco-Cruise Control mode where the vehicle detects road grade, calculates road's resistance and the acceleration is adjusted based on selected reference speed, current speed, road grade and vehicle's components internal behavior to save fuel. The use of GPS or topographic maps is not included.

Other studies, in simulation environment, prove that it's possible to save fuel based on road's topography [10, 11]. In [10], a high resolution map of road inclination, car's fuel consumption and dynamics models are used in a cost function to estimate the optimal speeds. Their solution for the optimal control problem it's addressed with three different methods, resulting in a fuel optimization of 5% compared to the conventional cruise control.

In [12], the energy efficiency is studied, considering not only the topographic data, but also the surrounding traffic flow in a combination of the ECC, ACC and V2V technology.

# Chapter 3

# Methodology

This chapter presents the methodologies adopted for data collection and analysis, considering the relation between the data to achieve the final results.

## 3.1 Hardware Development

For this study, a test platform and sensors were needed. The test car was a Smart Roadster (figure 3.1) that is equipped with an electronic throttle control (ETC). With ETC, the accelerator pedal is not connected directly to the throttle, but instead, the throttle actuator is controlled by an analog signal, which amplitude represents the pedal position.

Figure 3.1: Test platform.

### 3.1.1 Digital Accelerator Pedal

To take control over the throttle, a printed circuit board (PCB) was developed. This circuit is the key to make a stable prototype and safely bypass the accelerator output lines.

The accelerator connector to the engine control unit (ECU) is composed by six wires. Replacing the connector between the accelerator and ECU by a cable with probes and using a multimeter, it was identified the accelerator pinout, its internal circuit and even how ECU tests if the accelerator is working. The accelerator is composed by a pair of potentiometers, and each potentiometer is connected to different power lines, as showed on the circuit of figure 3.2. With the multimeter we could see that if $VCC1 == VCC2$, $OUT2 = \frac{1}{2}OUT1$, which allows us to use only one digital to analog converter (DAC) and obtain $D\_OUT$, which replaces the accelerator's $OUT1$ and the signal of the second output is obtained from $D\_OUT$ using a voltage divider.

The switching circuit has also probes on the accelerator outputs, connected to analog to digital converters, so that when the switch is commuted to use the digital accelerator, we can read the accelerator pedal signals amplitude and replicate them through oure digital circuit to the ECU. These microcontroller (MCU) inputs are also used to detect if the accelerator is pressed while the CC is turned on, making it turn off.

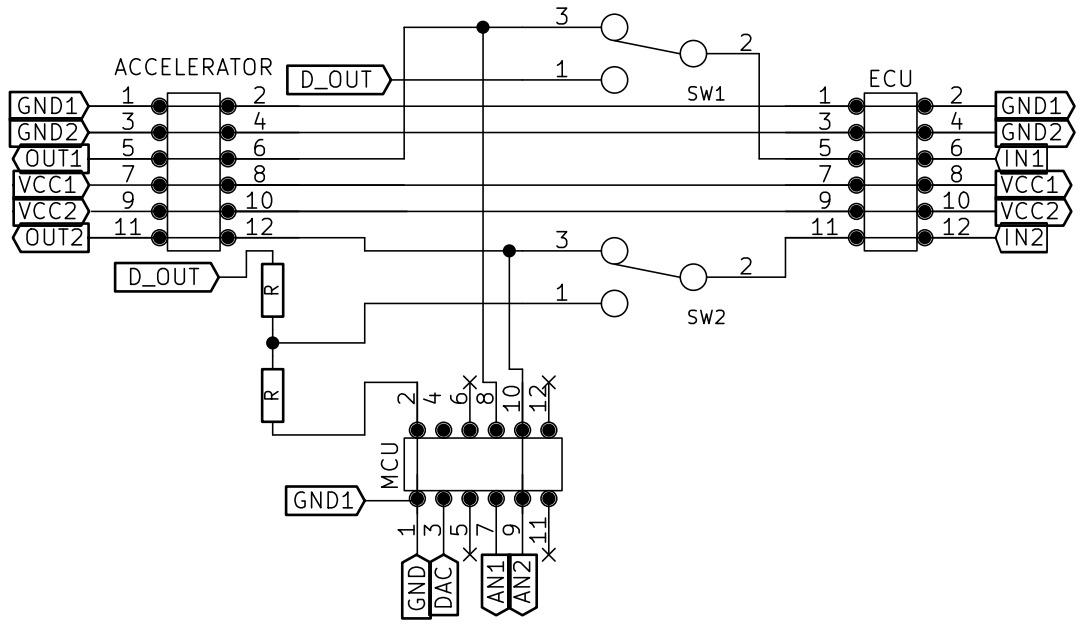



Figure 3.2: Circuit to connect MCU to the ECU and accelerator pedal. The switches SW1 and SW2 are actually only one switch, that is capable of switching two lines at the same time.

$$G_{ampop} = \frac{Accelerator_{vmax} - Accelerator_{vmin}}{DAC_{vmax} - DAC_{vmin}} \tag{3.1}$$

After measuring the output voltage of the accelerator output $OUT1$ for the minimum and full-throttle positions and the output voltage of the DAC for the digital value 0 and 4095, as shown on table 3.1, the circuit of fig 3.3 was developed, which allows us to use the full 12 bit DAC resolution and have a precise control over the throttle.

In normal accelerator operation, both $VCC$ lines are at $5V$ and $OUT2 = \frac{1}{2}OUT1$, but ECU is able to change the level of both lines independently, to verify if the accelerator is in good working

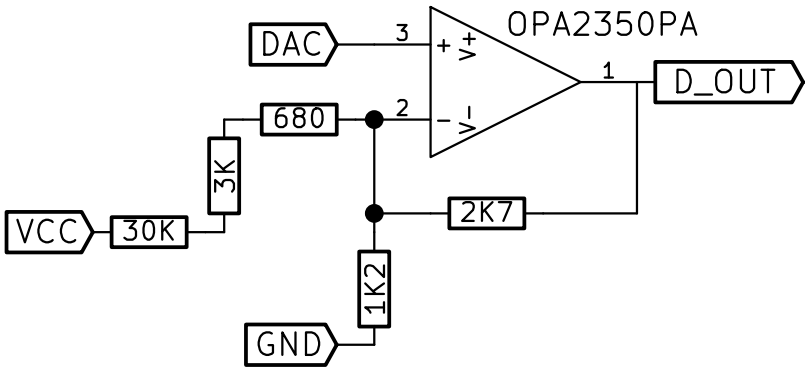



Figure 3.3: Amplifier circuit that adapts the DAC output range to the car's accelerator, it maps $[\min DAC_{out}, \max DAC_{out}] \rightarrow [\min Accelerator_{out1}, \max Accelerator_{out1}]$.

Table 3.1: Accelerator measurements.

| | Min (V) | Max (V) |
|---|---|---|
| Accelerator | 0.36 | 3.30 |
| DAC | 0.56 | 2,75 |
| DAC (mapped) | 0.36 | 3.31 |

order. The circuit using only one DAC to bypass the accelerator signal does not allow us to "respond" to the ECU "messages", but fortunately, the only situation detected that ECU does an accelerator check, is when we keep a steady output for some seconds, which doesn't happen during CC or DCC operation modes because there are always output variations at higher frequency.

### 3.1.2 OBD-II Interface

To access real-time car information like speed, RPM, MAF and engine load, it was used an OBD-II to bluetooth adapter, which has inside an ELEM327 chip, that communicates with car's sensors network using the *Keyword Protocol 2000*.

A *C++* library was developed, based on the ELEM327 datasheet, to manage the communication through a bluetooth serial connection to the OBD-II adapter. The library has some useful functions to check connection state and restart connection in case of anomalies are detected.

### 3.1.3 GPS

The GPS system is essential to this project but it brought a number of problems.

The GPS used was an Arduino shield (EB-5531), that comes with a water proof external antena and a SiRF Star IV chipset. In order to save time on the hardware development, we did try to use open source libraries to parse the GPS messages, but they showed to be to inefficient because:

- they use the NMEA protocol, that produces huge data through the serial port and is missing advanced options like select the GPS messages we want to receive;

- the reading routines are based on delays that block all system for intervals of one second long;
- data timestamp isn't calculated based on the GPS's one pulse per second (1pps) signal, being inaccurate.

To solve this problems, we:

- configured the SiRF Star IV chipset to use the SiRF Binary Protocol with baudrate 38400$bps$ (NMEA message "`$PSRF100,0,38400,8,1,0*3C`"), and configured the messages we want to receive (Measure Navigation Data Out – Message ID 2 and Geodetic Navigation Data – Message ID 41 of SiRF Binary protocol);
- read GPS serial data from buffer periodically, instead waiting for it inside a loop. It can be done because the maximum amount of data from the GPS, due to the selection of only two messages with 41 and 91$bit$, is now, $4 \times (16+15)+41+91 = 256 bit/s$ and the serial buffer has 512$bit$ capacity;
- developed *C++* library to parse the SiRF Binary protocol messages, detect transmission errors and associate messages to the respective *1pps* instant, taking advantage of the fact that the shield has access to the *1pps* pin, that rises a pulse every second, matching the instant when GPS data was calculated, and used the system's running time microseconds of that event to timestamp all GPS related data.

### 3.1.4 Barometric Pressure Sensor

The barometric pressure sensor used was the BMP180, that has an I2C interface. To communicate with the sensor, it was used the open source library "Adafruit_BMP085". This library includes functions that implement the communication protocol and to convert the barometric pressure to altitudes in meters.

The major problem of this sensor is that we don't get the same altitude value for the exact same spot if we read it in different days. The difference of pressure from one day to another have considerable impact on the altitude calculation. The other problem is that the sensor does not have a signaling mechanism. It sends data only when we ask for it. However, the pressure reading is not stable and its variations reflects on oscillations of 2 meters, so is not critical that we consider the time we ask for pressure data as the time when it was read.

### 3.1.5 MCU

Reached the point where we know all the sensors needed and the car's accelerator interface, we had to decide which MCU to use.

To facilitate the prototype development, we decided to choose a development board with a MCU that should have:

- three serial interfaces to communicate with bluetooth, GPS and database server;
- one I2C interface to communicate with the barometric pressure sensor;
- one DAC to override the accelerator;
- thirty available I/O pins to connect the LCD, Keypad, LEDs and hardware control pins;
- one I/O pin configurable as interrupt for the GPS 1pps.

The final choice was the Arduino Due development board because it comes with a *SAM3X8E* MCU, that covers all needs, it has a good Application Programming Interface (API) and it already has a serial to USB converter that is useful to communicate with a *Raspberry Pi*, where it will be running the server application.

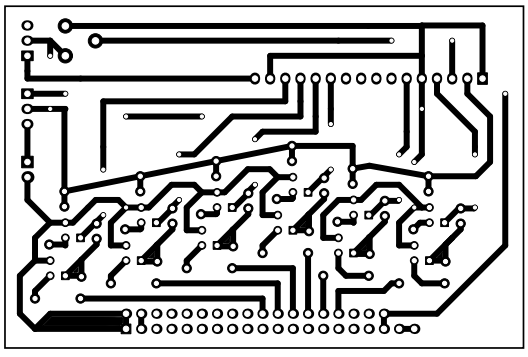
### 3.1.6 User Interface

To allow the driver to select/change the reference speed and some advanced configurations for development and testing, it was created a user interface that consists on a LCD display and a keypad board attached on a wood stick near the wheel for easy access while driving.

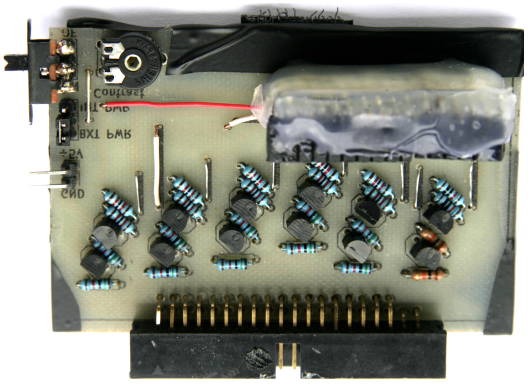
#### 3.1.6.1 LCD display

A LCD display is important to this project, because it can be used to show different menus such as to configure controller parameters, show debugging information and to display real-time sensors/server data.

The LCD control lines operate at $5V$ and the MCU I/Os, only operate at $3.3V$. To make the LCD work, a circuit was developed to convert all the LCD inputs to $5V$. This circuit board also has a jumper that allows us to use the Arduino $5V$ power supply or an external power source. The external power source is used, connected to car's lighter socket, so that we are not consuming to much current from the Arduino.



(a) LCD PCB etch cooper.



(b) LCD finished PCB.

Figure 3.4: LCD PCB project using Kicad software.

#### 3.1.6.2 Keypad

The keypad is another PCB, that was developed in order to introduce user inputs, and it's where the switch connected to the digital accelerator pedal board described in section 3.1.1 is incorporated.

The board integrates six pressure buttons with multiple purposes:

**UP/Down** it's used for menu navigation and for changing values on menus like select route or changing reference speed.

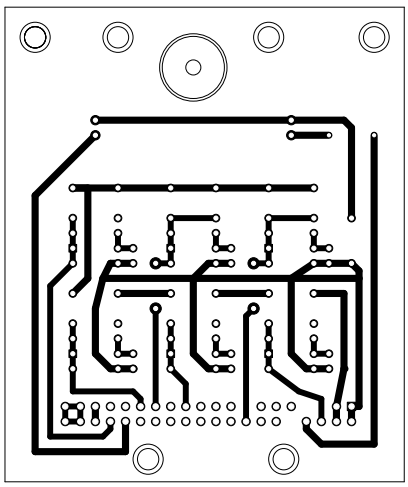
**Menu** it's used to change from the current menu to the next.

**OK** it's used to confirm an action or selection.

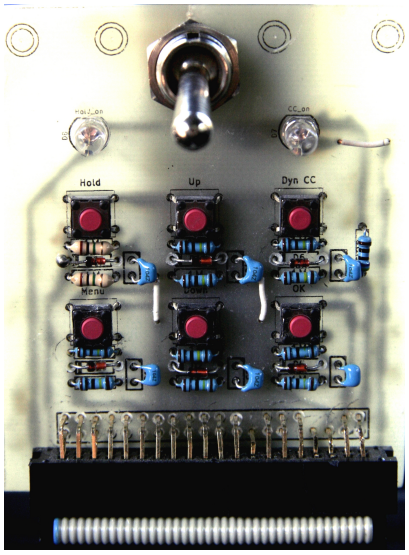
**Hold** it's used to signalize on the log files an event occurrence. This is useful, for example, to register experiments like the PI controller response to a step input and be able to filter them during data analysis.

**Dyn CC** it's used to activate/deactivate the CC/DCC, depending on the current selected operation mode. When the CC is activated, it sets as reference speed the actual car speed. It can be used also to resume the previous used reference, when pressed after the OK button, while on the main menu.

The LEDs shows to the driver if CC is on or off, as well as the hold function.



(a) Keypad PCB etch cooper.



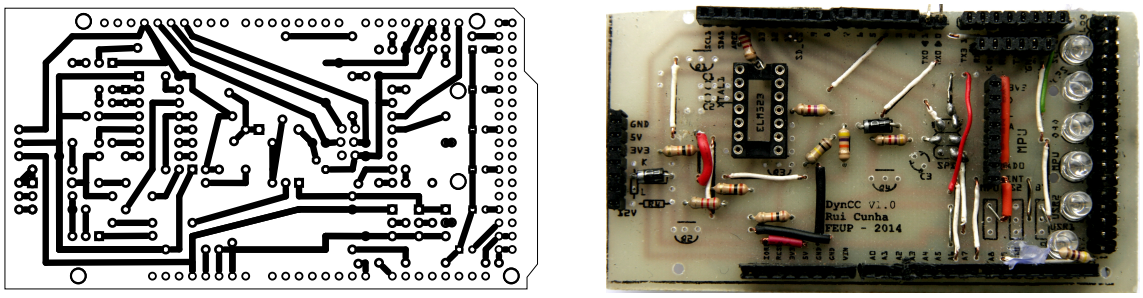

(b) User interface keypad.

Figure 3.5: Keypad PCB project using Kicad software.

### 3.1.7 Main Board

To connect all the components and PCBs, it was developed a board with the format of an Arduino Due shield. This board was projected with an option to connect to the OBD-II access port with a

cable, using a ELEM323 chip to communicate with car's KWP2000 protocol. However, it wasn't implemented, because the bluetooth solution is good enough to accomplish the desired results. It also have LEDs to show sensors and server communication activity.




(a) Main PCB etch cooper.

(b) Main PCB assembled.

Figure 3.6: Main PCB project using Fritzing software.


Figure 3.7: View of all system mounted on the car.

## 3.2 Data Logging

### 3.2.1 Data Acquisition

Due to the operational mode of the different sensors used, data needs to be stored with accurate timestamps. The GPS has an interrupt pin that indicates when GPS data is acquired, data ready on car's OBD-II diagnostics port is signalized with a feed character on the serial port and barometric pressure is acquired synchronized with GPS interrupt. Since the sampling frequency of the barometer is very high and there is a reasonable measuring error, it's not harmful to make the sensor reading at the instant of GPS interrupt, and take that timestamp to identify the reading. All data is logged on a CSV formated file named with current date and route identifier.

### 3.2.2 Data Synchronization and Pre-Prossessing

Before data is permanently stored on a SQL database, it has to be synchronized and some data is pre-processed to save real-time processing load. A Matlab script reads a log file at a time and for each file, it creates a vector with the timestamps of all data sources and then interpolates data from OBD-II to the GPS logging instants, since the remaining data is already synchronized.

The pitch angle is required to study vehicle dynamics and to be used in optional optimization targets. The same script also calculates the pitch angle of the road by implementing equation 3.4 as shown in figure 3.14 which illustrates the pitch angle calculation of an uphill situation.

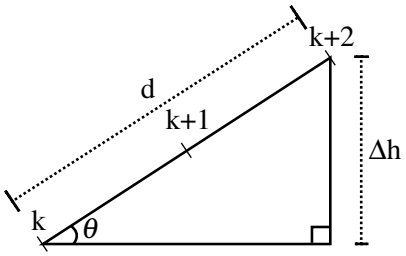



Figure 3.8: Slope angle calculation.

$$\Delta h = h(k+2) - h(k) \tag{3.2}$$

$$d = \frac{1}{2}(v(k) + v(k+2)) \times (t(k+2) - t(k)) \tag{3.3}$$

$$\theta(k+1) = sin^{-1}\left(\frac{\Delta h}{d}\right), \quad |\Delta h| < d \tag{3.4}$$

Finaly, the data processing script calculates the tangential acceleration based on vehicle speed and time.

$$a_t(k+1) = \frac{1}{2} \times \frac{v(k) + v(k+2)}{t(k+2) - t(k)} \tag{3.5}$$

### 3.2.3 Data Storage

Estimating the optimized speeds for a route, requires massive processing power, therefore, data storage system had to be organized thinking on minimizing the processing load, in trade of data storage space used.

As stated in 3.1.4, barometric pressure in the same place varies significantly from one day to another due to atmospheric changes. To avoid errors on the pitch angle calculation, data is stored with a trip identifier, so that we are able to calculate pitch angle separately, for each trip. To minimize the error from barometric pressure readings, we can calculate for each route, the average pitch over all trips.

To improve system's real-time performance, a database table was created to store, for each route, a map of optimal speeds, resultant from the fuel optimization algorithm. This table has a column to keep the reference speed used during the optimization process. With this strategy, we can optimize each route for a range of reference speeds and on the real-time system if the driver selects a speed for which there is no optimization done, we can use the average optimal speeds for the neighbor references, with a cost of optimization loss. This approach results in a very low real-time processing load, because the controller unit only need to query the database for a speed map based on selected reference speed, route id and vehicle's course and position.

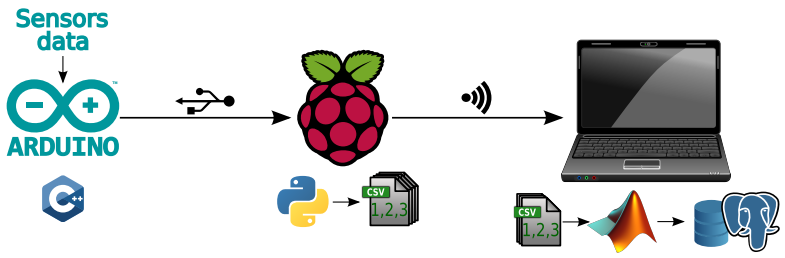



Figure 3.9: Data pre-processing and storage flow.

### 3.2.4 Data Pos-Processing

The database has tables to store data calculated based on all trips stored.

There were developed triggers and functions to update data in order to improve maps resolution and real-time performance:

- after upload, to the database, the last trip's log file of the waiting list, for each point of the route, update $\underset{all\_trips}{avg}(pitch)$, $\underset{all\_trips}{avg}(v)$ and $\underset{\min(dist)}{avg}(h)$, where $h$ is calculated using the inverse logic of the pitch in 3.2.2, but using the value of $\underset{all\_trips}{avg}(pitch)$. The function calculates, for each point $k$, the minimum geographical distances $(min(dist))$ between the point $k$ and all the other points, where $|\Delta COG| < 45^{o}$, using that set of data to calculate the averages.

- function "update_mpas", that for each route, order all data based on coordinates distances, starting from route's start coordinates, until route's end coordinates.

- function "update_zones_limit_speed", that every time that a new log file is stored, calculates average speed used on all points inside a speed limit zone of that route. This way we can force the optimization algorithm to use that speeds inside zones of speed limit like tolls, nodes and intersections, and the car will behave according to the user's driving style.

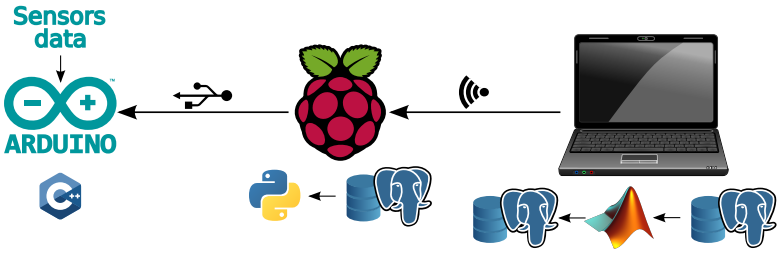
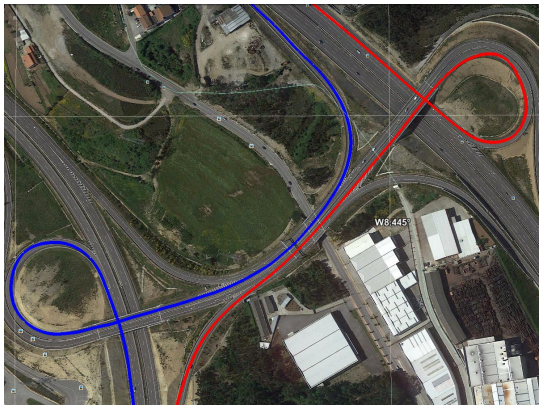



Figure 3.10: Data pos-processing flow.

If we store, for each route, terrain information and the average speeds used for each point, we save a lot of processing time. A Matlab script was developed to generate this maps, exporting at the end, a *kml* file that can be viewed using Google Earth, so that it's possible to check if the ordering algorithm works correctly, as shown on figure 3.11.

(a) Projection of all trips.



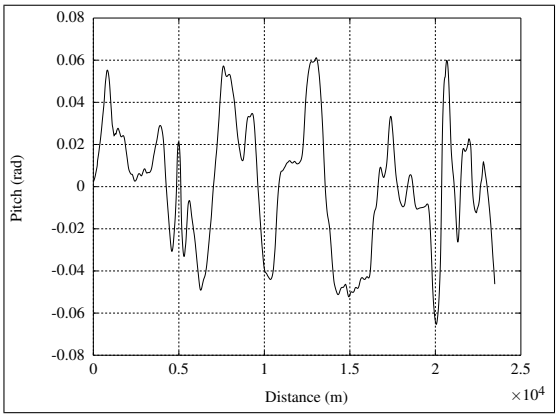
(b) Projection of routes after processed by coordinates ordering process, merging all trips into one single line.

Figure 3.11: Google Earth projections of two routes.

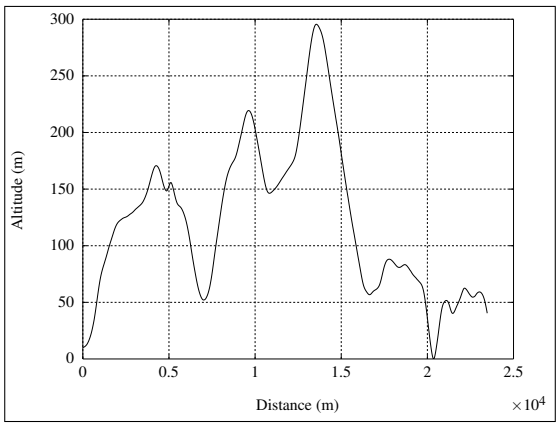
Using figure 3.12, we can also see if anything went wrong during the coordinates ordering process. If we take figure 3.12c, it's possible to see if there are zones with low resolution, like at the tunnel, where we have a step that corresponds to an impulse on $\Delta d$ has represented on figure 3.13.

### 3.2.5 DCC Data Server

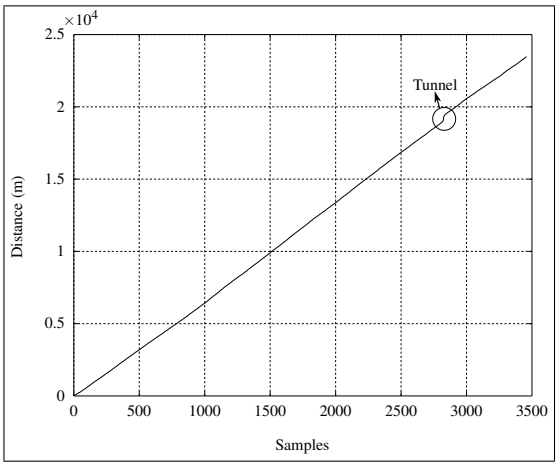
To implement the DCC, a Raspberry Pi was used, it holds the database and a server process, written in python, that establishes connection between the DCC process running on the MCU and the database, using a serial connection, through USB, and a simple communication protocol.



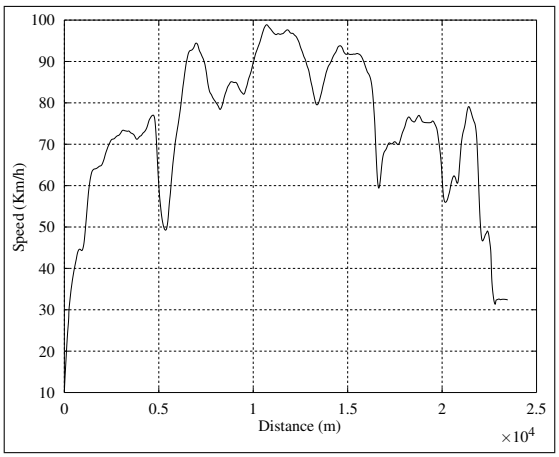

(a) Pitch angle.




(b) Relative altitude.




(c) Measured distance between ordered points.




(d) Overall trips average speed.

Figure 3.12: Plots of route 1, using the map with ordered coordinates.

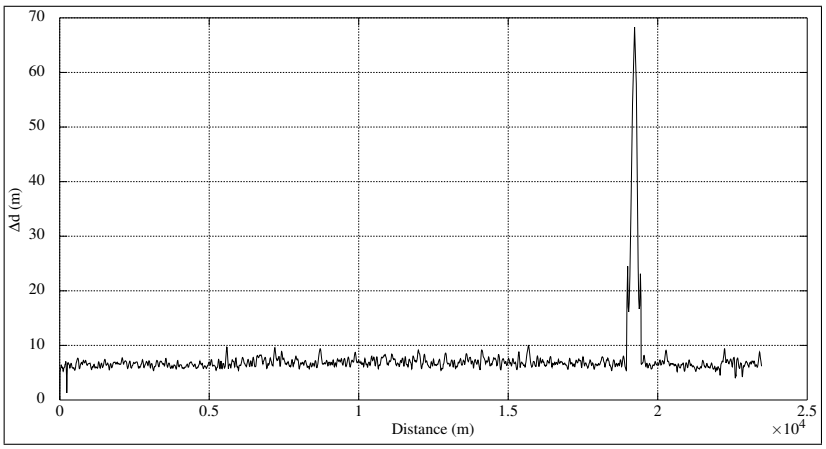



Figure 3.13: Route 1 map resolution.

The Raspberry Pi acts has a server, so it waits for Arduino messages, that are based on a remote procedure call (RPC) approach. After the handshake between the server and the client,

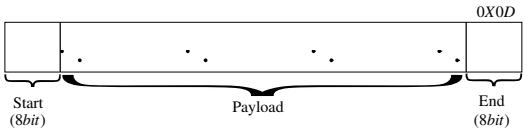



Figure 3.14: Communication protocol between Arduino and Raspberry Pi.

the messages from Arduino start with a character that identifies a function implemented on the python server code and the payload carries the function arguments and data. The server functions available are described on table 3.2.

Table 3.2: Server side functions available for RPC over the serial port.

| **Function** | **Start Byte** | **Function description** |
|---|---|---|
| $log\_data()$ | $0X61$ | Used to log sensors data to the trip data file. |
| $log\_system()$ | $0X62$ | For MCU system debugging, separated from the sensors data. |
| $get\_pitch(latitude, longitude, COG)$ | $0X63$ | Get pitch angle based on the average pitch of points inside the radius defined in $avg\_radius(m)$. $avg\_radius$ is not defined, it's calculated based on the points resolution of the current route. |
| $get\_speeds(latitude, longitude, COG, v_{ref})$ | $0X64$ | This function is similar to the $get\_pitch()$ with the difference that it only gets the average speed on a radius around the current location if there is not a optimized speed's map available for that location. If the route was already optimized, it will return the speed of the closest point on the speeds map, considering the selected reference speed. |
| $get_routes()$ | $0X66$ | This function queries the name and id for all available routes and returns the result. It is called from the MCU after a successful handshake and the user is prompt to select a route. |
| $set\_route(route_{id})$ | $0X66$ | Creates log files (system debug and trip data), identified with date and $route\_id$. If this function is called when a route was previously set, the log files are closed and new log files are created. |
| $set\_avg_radius()$ | $0X67$ | Manually override the radius, in meters, used to query for average data of points nearby. |
| $end\_trip()$ | $0X68$ | Terminates a trip, closing all files and stopping the logging functionality. |
| $exit\_app()$ | $0X7A$ | Calls function $end\_trip()$ and then sends a shutdown signal to turn off the Raspberry Pi. |

## 3.3 Control System

### 3.3.1 CC Controller

The CC controller is essentially a PI controller and it's the base of the controlling system, because it implements the conventional CC and it will be the plant of the DCC controller.

To project the PI controller, was taken as plant the system given by equation 3.6, that represents the car movement, where $m$ is the mass of the vehicle in $kg$ and $b$ is the damping coefficient in $Ns/m$. Using the PI controller $C(s)$ given by equation 3.7 and Matlab to produce the step response of the closed loop $H(s)$ (equation 3.8), the values of $K_p$ and $K_i$ were adjusted to obtain the smooth and steady response we desire. The value of $b$ is hard to calculate, so there were projected controllers for values of $b$ between $40Ns/m$ and $70Ns/m$.

$$P(s) = \frac{1}{m \times s + b} \tag{3.6}$$

$$C(s) = K_p + \frac{K_i}{s} \tag{3.7}$$

$$H(s) = \frac{C(s) \times P(s)}{1 + C(s) \times P(s)} = \frac{K_p s + K_i}{ms^2 + (K_p + b)s + K_i} \tag{3.8}$$

To fine tune the controller, it was developed a menu on the system's user interface that allows adjustments on the values of $K_p$, $K_i$, so that we could test the different controllers projected and than work on the one that is, in practice, closest to the expected result. The tunned controller has $k_p = 481$ and $K_i = 38.5$ and the step response of the transfer function $H(s)$ is represented on figure 3.16.

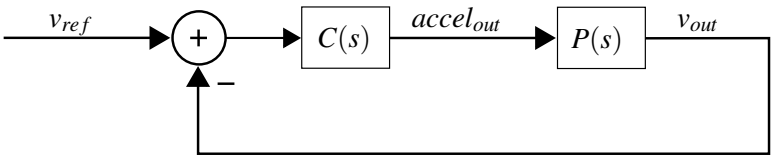



Figure 3.15: PI controller.

During the controller project, were considered weights of $m = 920Kg = m_{car} + m_{driver}$ and $m = 1500Kg$ (fig 3.17), making sure the controller is robust and can operate normally independently of the weight of the vehicle occupants and extra cargo.

The developed controller only operates on the accelerator pedal signal and not on the breaks, therefore, when we are on a downhill and the reference speed is crossed, the reference error integral value of the PI controller drives away very fast, trying to compensate the reference error. A similar problem occurs on a uphill where the car can't manage to keep the selected speed. This phenomena is known as *windup* and its problem is that the error integral value may drive away so far that when the terrain slope changes, making a normal control operation possible, the system will be "frozen", for an amount of time, equivalent to the time it was trying to compensate the reference error, in one of two states:

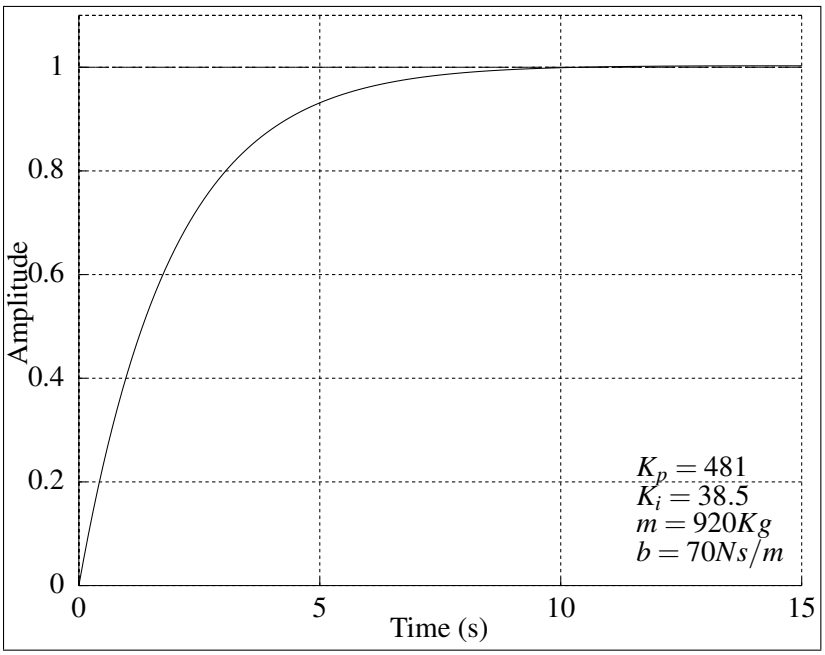



Figure 3.16: Final PI controller step response.

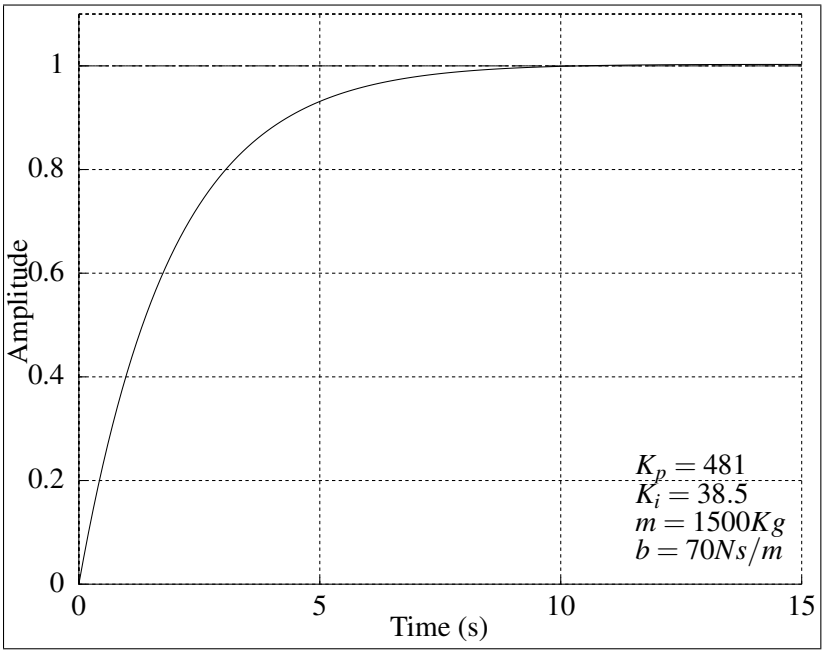



Figure 3.17: PI controller step response considering an exaggerated car weight.

1. If the *windup* occurs on a downhill, after that, the car almost stops until the integral stabilizes.

2. If the *windup* occurs on a uphill where increasing the accelerator will not give more speed, after that, the car will accelerate at its maximum, which is obviously dangerous.

To work around this *windup* situation, the implementation was changed to update the integral value only under some conditions represented on the following pseudo-code:

$windup = ((output(k-1) > accelerator\_in\_max)$ **and** $(output(k-1) \geq output(k-2))$
**and** $(speed(k) < speed_{ref}(k)))$ **or** $((output(k-1) < accelerator\_in\_min)$ **and**
$(output(k-1) \leq output(k-2))$ **and** $(speed(k) > speed_{ref}))$;
**if *not*** *windup* **then**
  compute error and integral;
**end**
**else**
  keep last reference error and integral values;
**end**

**Algorithm 1:** Anti-windup.

### 3.3.2 DCC Controller

Concluded the implementation of the traditional CC, we were in conditions to develop a controller that will set the reference speed used by the PI controller.

As stated in section 3.2.3, the energy optimization is made off-line and stored on the database, therefore, the speed controller is essentially based on a table lookup, using vehicle's current position and selected reference speed. The vehicle position is given by latitude, longitude and course over ground ($cog$), that indicates the direction of movement.

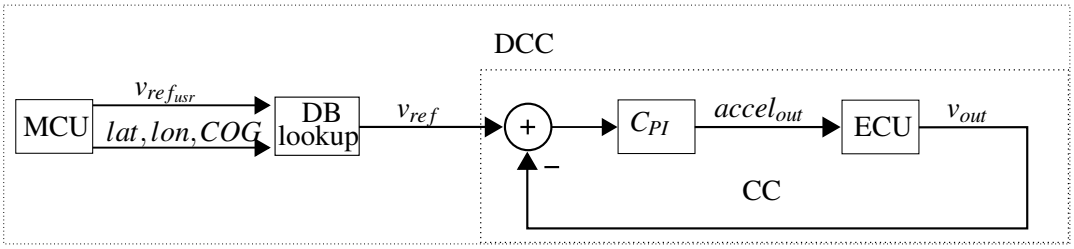



Figure 3.18: General view of the control loop.

## 3.4 Optimization Strategy

Optimizing fuel consumption for a route is a demanding task, so we simplified the problem.

At start, the idea of optimization was based a car's fuel consumption model, which implied a study of the relationships between RPM, MAF, gear position, engine load and road's pitch angle, until we get equations to estimate MAF and engine load based on pitch and RPM and then minimize $\sum \Delta fuel\_spent$ for that route. We can get RPM from speed using the $gear_{factor}$ as: $RPM = gear_{factor} \times v$. In this case the optimization algorithm would have to minimize the fuel consumption experimenting with both $v$ and $gear_{factor}$. The problems of this solution are:

1. The high complexity of the optimization algorithm. The optimization process would have to, at each iteration, experiment with values for speed and gear and calculate, based on

engine variables, the result of a fuel cost function. Making an optimization like this would take forever to process.

2. Not very interesting to the real world implementation, because it is strongly dependent on car's engine properties.

Then there was the idea of minimize the total energy the engine has to produce on a given route, respecting a set of restrictions. This results in a much simpler optimization process, that only depends on the car's drag force and road's altitudes and distances. Although the total fuel consumption of a route depends of the engine, if we optimize the energy, maybe, it's possible to achieve fuel savings near to the result of a fuel optimization function.

### 3.4.1 Cost Function

Any object with mass, $m$, at a velocity, $v$, with an altitude $h$, at a given instant $k$, has a mechanical energy calculated with the equation 3.11.

$$E_P(k) = g \times m \times h(k) \tag{3.9}$$

$$E_C(k) = \frac{1}{2} \times m \times v^2(k) \tag{3.10}$$

$$E_M(k) = E_P(k) + E_C(k) \tag{3.11}$$

The car has mechanical energy, but also is subject to drag force, so there is a drag energy ($E_{drag}$).

The drag force is calculated using equation 3.12, where $c_d$ is the drag coefficient and $A$ is the vehicle frontal area in meters. Starting from equation 3.12, we can deduce the drag energy.

$$F_{drag}(k) = \frac{1}{2} \times \rho \times c_d \times A \times v^2(k) \tag{3.12}$$

$$P_{drag}(k) = \frac{1}{2} \times \rho \times c_d \times A \times v^3(k) \tag{3.13}$$

$$E_{drag}(k) = P_{drag}(k) \times \frac{d(k)}{v(k)} \tag{3.14}$$

$$= \frac{1}{2} \times \rho \times c_d \times A \times d(k) \times v^2(k) \tag{3.15}$$

$$\tag{3.16}$$

If we can calculate $E_M$ and $E_{drag}$, the total energy is given by equation 3.17.

$$E_T(k) = E_M(k) + E_{drag}(k) \tag{3.17}$$

At this point, the goal was, for each route, use the information stored on the maps table and minimize $\sum \Delta E_T(k)$.

The optimization algorithm only have to try different speeds for each point of the altitudes map. If we look at equations 3.19 and 3.21, we see that to minimize the energy, the immediate answer is $v(k) = 0, k = 1 : N$.

$$\Delta E_P(k) = g \times m \times (h(k+1) - h(k)) \tag{3.18}$$

$$\Delta E_C(k) = \frac{1}{2} \times m \times (v^2(k+1) - v^2(k)) \tag{3.19}$$

$$\Delta E_M(k) = \Delta E_P(k) + \Delta E_C(k) \tag{3.20}$$

$$\Delta E_{drag}(k) = \frac{1}{2} \times \rho \times c_d \times A \times (d(k+1) - d(k)) \times \left( \frac{v(k+1) + v(k)}{2} \right)^2 \tag{3.21}$$

$$\Delta E_T(k) = \Delta E_M(k) + \Delta E_{drag} \tag{3.22}$$

So to force the algorithm to converge to a valid array of speeds, we have to add some restrictions, as follows:

- The average speed of the calculated array, must be as near as possible from the reference speed;
- The speed inside a zone of speed limit, has to be as close as possible from the user's habitual speed on that zone;
- The maximum speed can not be higher than the maximum defined;
- The minimum speed should never be lower than the minimum defined;

To implement the mentioned restrictions, we have to quantify each violation and use the results as weight factors of the cost function. The violations are then quantified by the next equations:

$$v_{ref_{error}} = (avg(v(k)) - v_{ref})^2 \quad : \quad k \supset zone_{limit} \tag{3.23}$$

$$v_{limit_{err}} = \sum (v(k) - v_{limit}(j))^2 \quad : \quad k \subseteq zone_{limit} \tag{3.24}$$

$$v_{min_{err}} = \sum (v_{min} - v(k))^2 \quad : \quad v(k) < v_{min} \quad \wedge \quad k \supset zone_{limit} \tag{3.25}$$

$$v_{max_{err}} = \sum (v(k) - v_{max})^2 \quad : \quad v(k) > v_{max} \tag{3.26}$$

$$\tag{3.27}$$

The result of the cost function is given by equation 3.29.

$$E_{Total} = \sum \Delta E_T(k) \tag{3.28}$$

$$F_1 = E_{Total}^2 + k_1 \times v_{avg_{err}} + k_2 \times v_{limit_{err}} + k_3 \times v_{min_{err}} + k_4 \times v_{max_{err}} \tag{3.29}$$

$F_1$ is then minimized using the Matlab function $fminsearch()$, using, as initialization, the average speeds practiced on the route, as represented on figure 3.19.

The weights of violations being adjusted, changes the optimization priority to focus not only on minimize the energy spent.

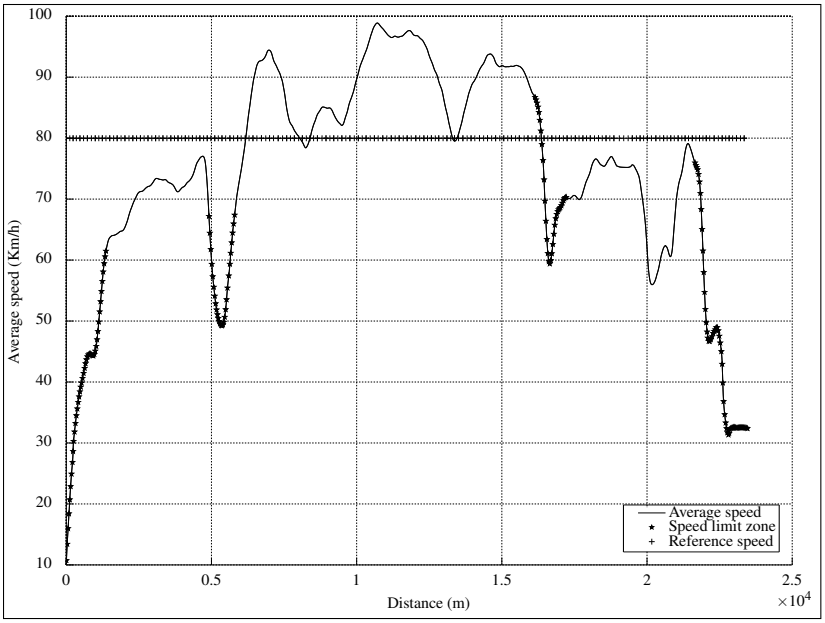



Figure 3.19: Overall trips, average speed, for the route to optimize, with representation of zones of limit speed.

The optimization process was applied for one of the two routes where the tests where performed. At first it took a long time to run, because of two problems. First, the map used had a resolution of $6m$, which results in 3458 points to optimize one route of $23.7km$. The second reason was a bad definition of the cost function, where the errors weren't multiplied by a constant $k_1, k_2, ..., k_n$, but by the total energy (equation 3.28), that is a variable, which difficults the optimization task and it's not a suitable solution for weights adjustments.

Using Matlab's function $downsample()$, we were able to reduce the map resolution to $380m$ instead of $6m$. The resolution was adjusted to the minimum possible before degrading the altitudes map information. Than, minimizing the result of $F_1$ for a small number of iterations, did help to adjust the weights of each violation according to the results of each error variable at the end of the process.

There were tested different weights $k_1, k_2, ..., k_n$ on the cost function $F_1$ in order to minimize energy, respecting all constraints, resulting on cost functions $F_{1_1}$ and $F_{1_2}$. On table 3.3 are represented the optimization results for two referred cost functions with the following weights respectively:

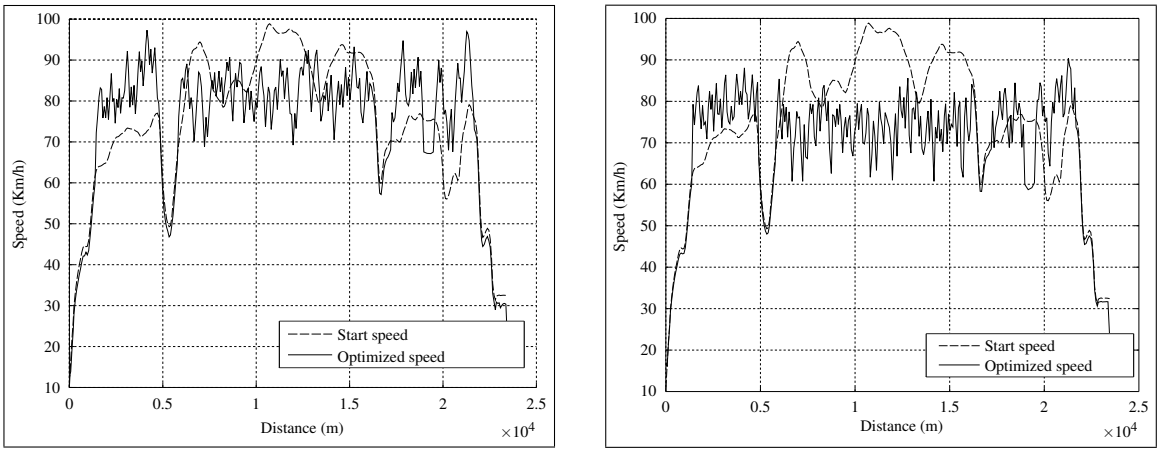
- $$k_{1_1} = 10^{12}, \quad k_{2_1} = 10^{10}, \quad k_{3_1} = 10^{6}, \quad k_{4_1} = 10^{11}$$

- $$k_{1_2} = 10^{12}, \quad k_{2_2} = 10^{10}, \quad k_{3_2} = 10^{5}, \quad k_{4_2} = 10^{10}$$

Table 3.3: Comparison of optimization parameters taking the average speeds used, over all trips, as start of the iterative process, and the speeds resultant from the optimization process for route 1.

| | Before Optimization | After Optimization | |
|---|---|---|---|
| | | $F_{1_1}$ | $F_{1_2}$ |
| $v_{ref_{err}}$ | 0.00 | 0.19 | 1.96 |
| $v_{limit_{err}}$ | 0.00 | 43.54 | 20.56 |
| $v_{min_{err}}$ | 72.63 | 0.02 | 0.56 |
| $v_{max_{err}}$ | 0.00 | 0.00 | 0.00 |
| $a_{t_{err}}$ | 0.01 | 0.85 | 4.06 |
| $E_{Total}$ | $5037.40 \times 10^3$ | $4320.06 \times 10^3$ | $4259.22 \times 10^3$ |

Looking to the optimization results for $F_{1_1}$ we can conclude that we got a small energy reduction of 0.63%. So in $F_{1_2}$, the weights were adjusted to focus on the energy optimization. The minimization of $F_{1_2}$ did indeed result in an energy reduction of 13.50%, but the average speed was $5Km/h$ below the reference speed. To test new values for the cost function weights it's not easy, specially because of the time that the process requires to execute. The other problem with the cost function $F_1$ is that the resultant speeds doesn't make sense, because they oscillate too much, has we can see on figure 3.20.




(a) $F_{1_1}$. (b) $F_{1_2}$.

Figure 3.20: Optimal and initial speeds comparison using cost function $F_{1_1}$ and $F_{1_2}$.

Something was missing on the cost function, to avoid the speed variations and force the reference speed to be respected while saving energy. To stabilize the reference speed error near zero, allowing a wider window for optimization evaluations, we added an extra average speed error penalty $v_{ref_{err2}} = |avg(v(k)) - v_{ref}|$. The speed variation problem, was solved by adding an acceleration constraint, $a_{t_{reduc}}$, that results in an overall acceleration reduction. The output is represented in figure 3.21.

Upon this results, we can see that the optimization process still had problems to use the available speeds range. This could be corrected by tunning the weights we already had defined, but we

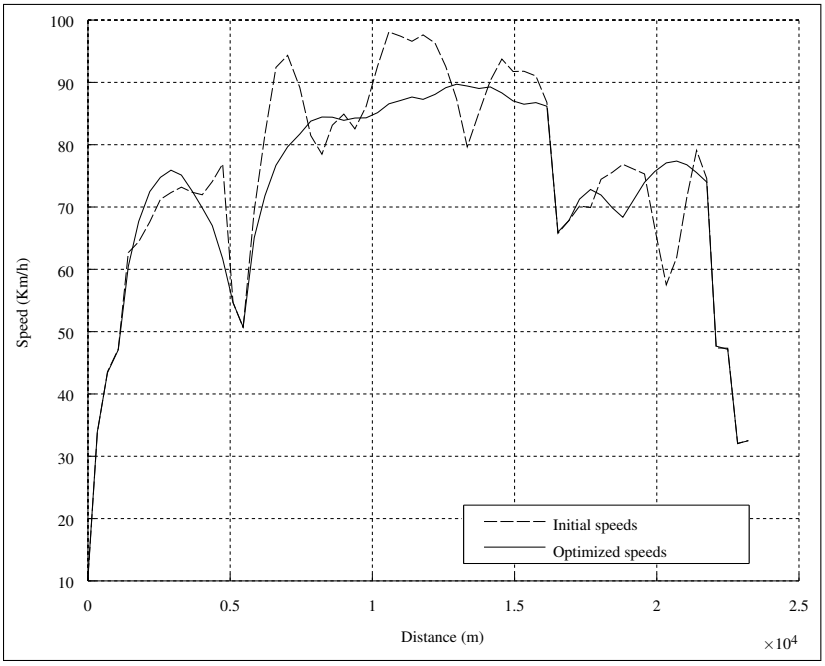



Figure 3.21: Optimized speeds with acceleration limited.

decided to add one last term to the cost function, that introduces a tendence to achieve the higher speeds at the end of a downhill and the lower speeds at the top of a uphill. This restriction helps the optimization process to converge in less than a minute. To identify the top and bottom of road's hills, it was used the following procedure:

1. Calculate the altitude differences $\Delta h(k) = h(k+1) - h(k)$;
2. Calculate the signal of $\Delta h$, which results on the values $0, 1, -1$ in case it is a flat surface, an uphill or a downhill, respectively;
3. Calculate the differences, $changes = sig(\Delta h(k+1)) - sig(\Delta h(k))$, which results on the values $0, 2, -2$;
4. Obtain top of the hills as, $changes == 2$ and the botom of the hills as, $changes == -2$;
5. Finally, we get the variables, $h_{max_{err}} = \sum (v(k) - v_{min})^2$, with $k \subseteq hills_{top}$ and $h_{min_{err}} = \sum (v(k) - v_{max})^2$, with $k \subseteq hills_{botom}$.

The final cost function is then given by equation 3.30.

$$F_2 = E^2_{Total} + k_1 \times v_{ref_{err}} + k_2 \times v_{ref_{err2}} + k_3 \times v_{limit_{err}} + k_4 \times v_{min_{err}} + k_5 \times v_{max_{err}} + \\ + k_6 \times a_{t_{reduc}} + k_7 \times h_{max_{err}} + k_8 \times h_{min_{err}} \tag{3.30}$$

## 3.5 Fuel Consumption Measurements

To obtain reliable fuel consumption measurements, we did an effort to make sure that:

- Every time the fuel tank was low, was used the same gasoline pump and the tank was always completely filled. At that instant, the car's odometer was reset to zero and it was taken note of the kilometers done since last refill, the liters we did add to the tank and the driving method used.

$$Fuel_{med} = \frac{Fuel_{in}}{Distance} \times 100 \quad (L/100Km)$$

- The car was used always on the same traject with two way routes mostelly highway and between each fuel refill, only one driving method was used.

- The highway average speed was kept with the value of $80Km/h$ for all driving methods.

With this method, we can accurately measure fuel consumptions to establish relationships between each driving method.

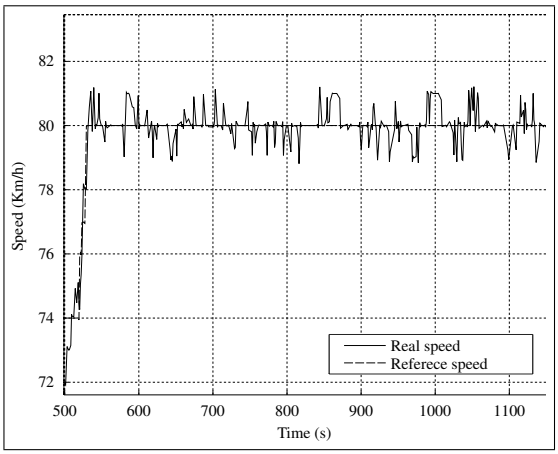
# Chapter 4

# Measurements

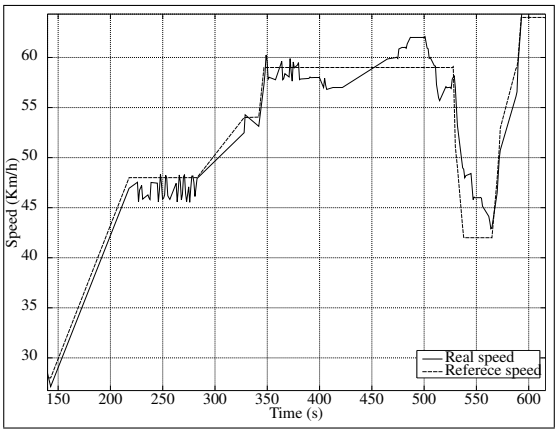
## 4.1 Control System

### 4.1.1 PI Controller

In practice, the controller works has expected. That can be proved from figure 4.1, that is the plot of data collected while using the developed cruise control and it shows that the vehicle speed always follows the reference speed.




(a) Following a constant reference.




(b) Following a variable reference.

Figure 4.1: PI controller performance on the road.

In figure 4.2, is represented the step response to a degree reference change. The result is a fast, yet smooth response, a slight overshoot, but a quick stabilization.

### 4.1.2 DCC

Has the system is aware of speed limit zones, and the speeds map is filtered, it delivers a smooth driving experience, reducing the driver's task to control the steering wheel. The breaks are only

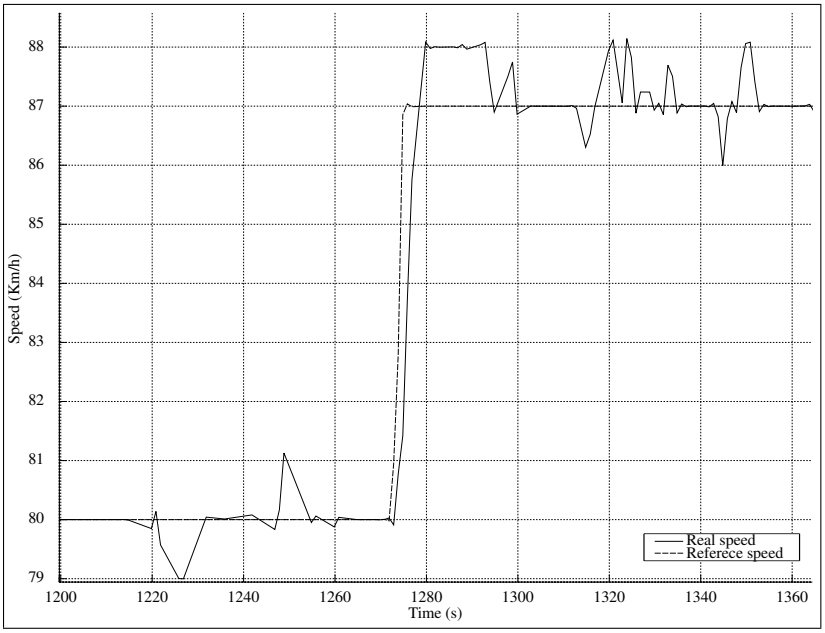



Figure 4.2: PI controller real step response.

used if the traffic forces it to happen. It is possible to confirm these results with figure 4.3, that compares the reference speed set by the server running on the Raspberry Pi and car's real speed. There are some overshoots and undershoots that are caused by the windup phenomena, that wasn't correctly eliminated.

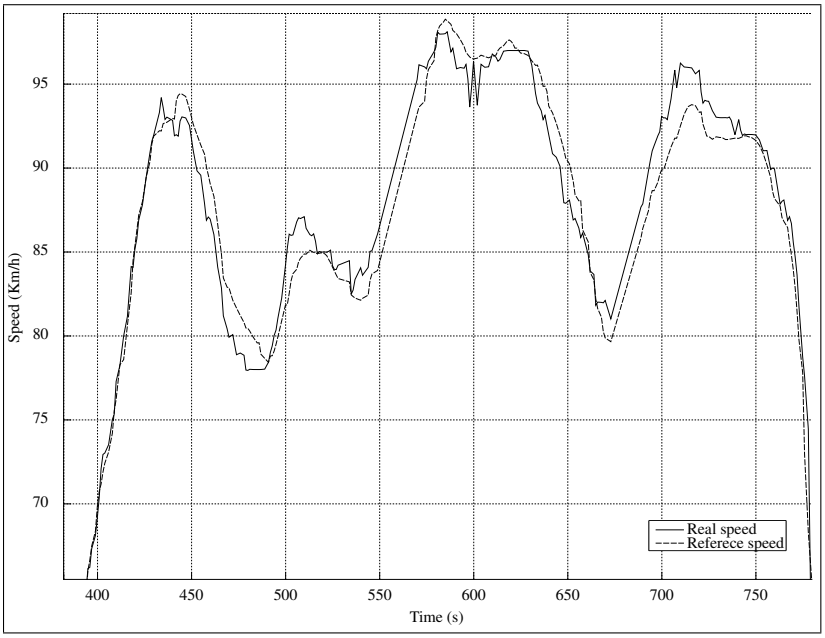



Figure 4.3: Comparison between server's planned speeds and car's execution.

## 4.2 Cost Function and Optimization Algorithm

The cost function $F_2$ from equation 3.30, was minimized using the altitude maps created on the pos-processing phase and shows to converge after 9000 iterations of the Matlab's *fminsearch* function. Table 4.1, shows the optimization process results for route 1 (23.7$km$).

Table 4.1: Comparison of optimization parameters taking the average speeds used, over all trips, as start of the iterative process, for route 1, using cost function $F_2$.

| | Before Optimization | After Optimization |
|---|---|---|
| $v_{ref_{err1}}$ | 0.00 | 0.21 |
| $v_{ref_{err2}}$ | 0.00 | 39.99 |
| $v_{limit_{err}}$ | 0.00 | 0.46 |
| $v_{min_{err}}$ | 0.69 | 0.00 |
| $v_{max_{err}}$ | 0.21 | 0.00 |
| $a_{t_{reduc}}$ | 0.75 | 0.53 |
| $h_{max_{err}}$ | 227.34 | 31.17 |
| $h_{min_{err}}$ | 472.05 | 432.51 |
| $E_{Total}$ | $5037.40 \times 10^3$ | $4854.84 \times 10^3$ |

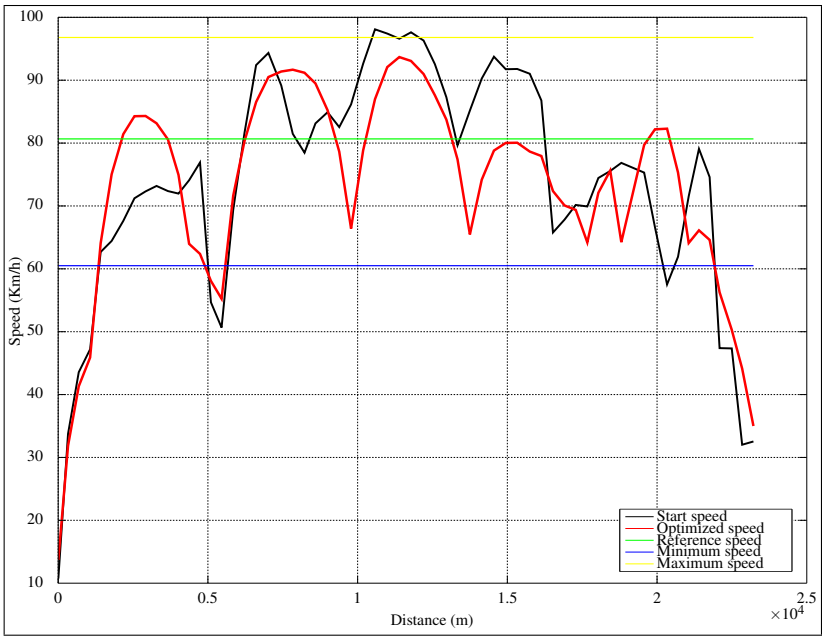



Figure 4.4: Optimal and initial speeds comparison for route 1 using cost function $F_2$.

The cost function $F_2$ was also tested with route 2 (23.99$km$) and we got the results of table 4.2

In a real application, the optimization process should converge from anny initial speed array, because we don't have the average speeds of every route. The algorithm was tested for a initial speed array equivalent to the use of the traditional CC on route 1, respecting the limit speed zones, which for the same number of iterations used on the previous tests shows to converge to a similar result, represented on figure 4.6.

Table 4.2: Comparison of optimization parameters taking the average speeds used, over all trips, as start of the iterative process, for route 2, using cost function $F_2$.

| | Before Optimization | After Optimization |
|---|---|---|
| $v_{ref_{err1}}$ | 0.00 | 0.21 |
| $v_{ref_{err2}}$ | 0.00 | 0.46 |
| $v_{limit_{err}}$ | 0.00 | 85.60 |
| $v_{min_{err}}$ | 0.69 | 0.86 |
| $v_{max_{err}}$ | 0.21 | 0.00 |
| $a_{t_{reduc}}$ | 0.75 | 0.75 |
| $h_{max_{err}}$ | 227.34 | 39.74 |
| $h_{min_{err}}$ | 472.05 | 891.90 |
| $E_{Total}$ | $5037.40 \times 10^3$ | $3908.59 \times 10^3$ |

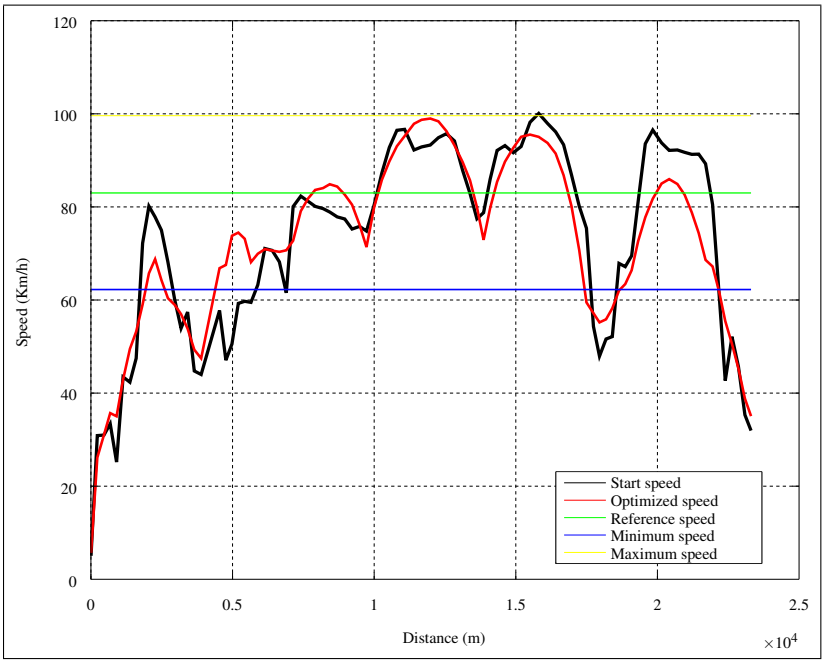



Figure 4.5: Optimal and initial speeds comparison for route 2 using cost function $F_2$.

Testing the optimization algorithm, replacing the elevation profile by a flat surface, we can see that the result is similar to a traditional CC. The reason it doesn't result on a constant speed is the need to keep the average speed when it is not allowed to accelerate instantly to the reference speed after a speed limit zone.

## 4.3 Fuel Optimization

Before the optimization process was ready, it was tested a manual application of the expected result to estimate the potential savings it could produce and to ensure some results in case we didn't have time to accomplish the automatic system.

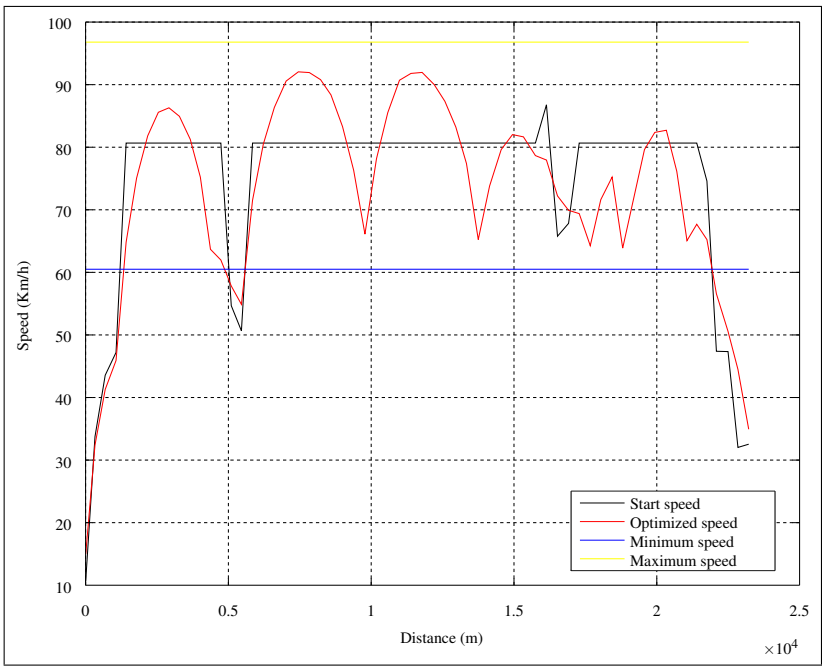



Figure 4.6: Optimization result starting from a constant speed.

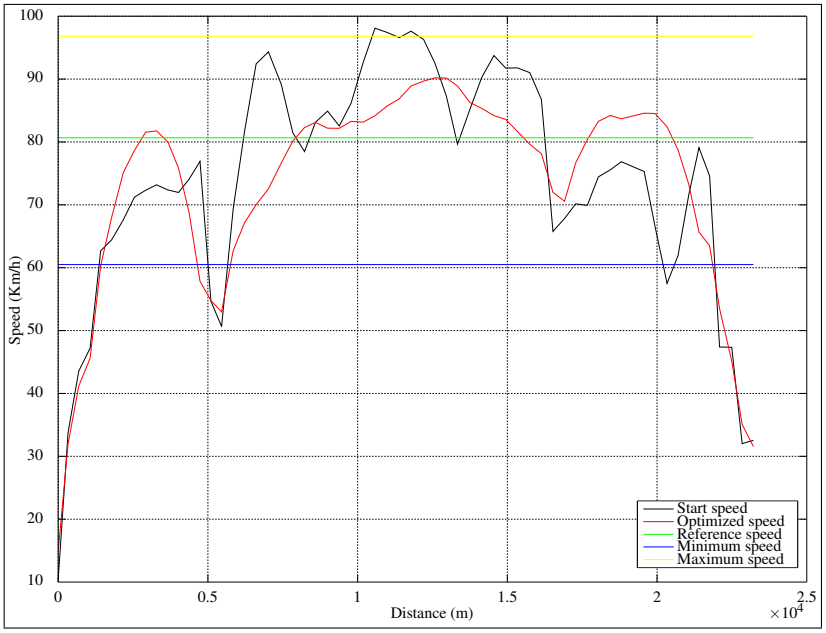



Figure 4.7: Optimization result for a flat road.

The manual application consists on the use of normal CC, but the reference speed was gradually changed based on personal road knowledge, using, in practice, a refined version of algorithm 2.

Looking to the algorithm 2, it seems to be easy to implement, but in reality, the speed changes made are a lot more complex than this, because we can't just separate downhills from uphills and flat ground. To do a real optimization we need to consider the slope angle variation over each

**if** *downhill* **then**
  gradually increment reference speed;
**end**
**else if** *uphill* **then**
  gradually decrement reference speed;
**end**
**else**
  set speed = reference speed;
**end**

**Algorithm 2:** DCC (manual mode).

scenario and adjust the speed at different rates. It is not hard to do manually when we know both car and the road well enough.

This process proved that it is possible to reduce significantly the fuel consumption, as shown on table 4.3, keeping the same average speed. By logging the mileage and gasoline liters spent for each fuel tank, we did observe a fuel consumption reduction of 10.74% comparing with the traditional CC and 15.92% compared to manual driving.

Has expected, the full implementation of the DCC did outbid its manual test, resulting on a 23.25% fuel consumption reduction compared to the traditional CC for the same route and speed.

Table 4.3: Fuel consumption comparison using different driving methods.

| Driving Method | 98 Octane Gasoline (L) | Distance (Km) | Fuel Consumption (L/100Km) | | |
|---|---|---|---|---|---|
| | | | Max | Min | Avg |
| Manual | 249.14 | 4544.70 | 5.74 | 5.31 | 5.48 |
| CC | 85.46 | 1655.10 | 5.30 | 5.00 | 5.16 |
| DCC (Manual) | 85.18 | 1848.10 | 4.96 | 4.56 | 4.61 |
| DCC | 27.54 | 694.70 | 3.96 | 3.96 | 3.96 |

# Chapter 5

# Conclusions and Future Work

## 5.1 Results Discussion

The results from the fuel consumption tests were better than the expected, because they were higher than simulations of the ECC done in other studies, as stated in chapter 2 and we know that the results obtained can be even better, as the tests were made with the first working version of the optimization algorithm, that still has some bugs and adjustments to be made. The major problem of the current optimization applied on the car, is that sometimes we get a great accelerations in zones it wasn't suppose to happen because the limit speed zones were defined with incorrect area, and the anti-windup solution on the PI Controller is not optimal.

The DCC system developed shows to be capable of saving fuel compared to the traditional CC at any road's elevation profile, however, this fuel economy grows with the number of hills. Driving in a flat road will make no difference to use a DCC or a CC and both achieve the maximum fuel efficiency.

## 5.2 Future Work

The work developed is just the tip of the iceberg. Even though we are already able to save considerable fuel, there are adjustments and structural changes that should be made:

1. The cost function can be improved, by developing an optimization process to define the weights.

2. New optimization approaches can be developed, as optimize a route by partitioning it in smaller fractions based on slope angle interfaces, since the speed used in a point $A$ will not affect the consumption in $B$ if $A$ and $B$ are separated by several slope changes.

3. Optimize a route without the need of planning routes. Instead, using external maps sources and calculating possible road intersections, the optimization algorithm can run based on current position, COG and next road crossing;

4. Combining the last two improvements, the optimization process will converge so much faster that it's possible to migrate to real-time optimization;

5. The vehicle model can be refined by adding the road friction coefficients $c_0$ and $c_1$, by adding to the energy equation these parameters as a function of road slope and vehicle speed: $c_0(\theta) \times v + c_1(\theta) \times v^2$;

6. Combine the DCC with ACC to achieve better results on traffic and increase security and driving comfort;

After all mentioned changes, the developed device can be factory implemented on car's ECU without great effort to make it work to any car model, since the only thing we need to change is the car's drag coefficient and mass.

# References


[1] W. Levine and M. Athans. On the optimal error regulation of a string of moving vehicles. *IEEE Trans. Automat. Contr.*, 11(3):355–361, 1966.

[2] PA Ioannou and CC Chien. Autonomous Intelligent Cruise Control. *Vehicular Technology, IEEE ...*, 1993.

[3] Jian Wang, Xin Xu, Daxue Liu, Zhenping Sun, and Qingyang Chen. Self-Learning Cruise Control Using Kernel-Based Least Squares Policy Iteration. 22(3):1078–1087, 2014.

[4] Bart Van Arem, Cornelie JG van Driel, and Ruben Visser. The impact of cooperative adaptive cruise control on traffic-flow characteristics. *Intelligent Transportation Systems, IEEE Transactions on*, 7(4):429–436, 2006.

[5] LC Davis. Effect of adaptive cruise control systems on traffic flow. *Physical Review E*, 69(6):066110, 2004.

[6] Thomas Stanger. A Model Predictive Cooperative Adaptive Cruise Control Approach. 2013.

[7] Dominik Lang, Thomas Stanger, and Luigi del Re. Opportunities on fuel economy utilizing v2v based drive systems. Technical report, SAE Technical Paper, 2013.

[8] Fazal Urrahman Syed, Matthew Allen Warner, Ryan J Skaff, Benjamin Carl Mukkala, Terry Gene Feldpausch, Ming Lang Kuang, David H Schmitt, and Elaine Y Chen. Eco-mode cruise control, August 30 2013. US Patent App. 14/015,033.

[9] B Saerens, H A Rakha, and M Diehl. Eco-Cruise Control for Passenger Vehicles: Methodology. 2013.

[10] Sangjun Park, Hesham Rakha, Kyoungho Ahn, Kevin Moran, Bart Saerens, and Eric Van Den Bulck. Predictive ecocruise control system. *Transportation Research Record*, n 2270:113–123, January 2012.

[11] P Gáspár and B Németh. Design of adaptive cruise control for road vehicles using topographic and traffic information. pages 4184–4189, 2014.